**品优购电商系统开发**

**第20章**

**微信扫码支付与秒杀**

传智播客.黑马程序员

# 1.品优购-检测支付状态

## 1.1需求分析及实现思路

### 1.1.1需求分析

当用户支付成功后跳转到成功页面



当返回异常时跳转到错误页面



### 1.1.2实现思路

我们通过HttpClient工具类实现对远程支付接口的调用。

接口链接：https://api.mch.weixin.qq.com/pay/orderquery

具体参数参见“查询订单”API, 我们在controller方法中轮询调用查询订单（间隔3秒），当返回状态为success时，我们会在controller方法返回结果。前端代码收到结果后跳转到成功页面。

## 1.2检测支付状态-后端代码

### 1.2.1服务接口层

在interface项目的WeixinPayService.java中新增方法定义

|  |
| --- |
| /\*\*  \* 查询支付状态  \* **@param** out\_trade\_no  \*/  **public** Map queryPayStatus(String out\_trade\_no); |

### 1.2.2服务实现层

在buyer-service项目的WeixinPayServiceImpl.java中实现方法

|  |
| --- |
| @Override  **public** Map queryPayStatus(String out\_trade\_no) {  Map param=**new** HashMap();  param.put("appid", appid);//公众账号ID  param.put("mch\_id", partner);//商户号  param.put("out\_trade\_no", out\_trade\_no);//订单号  param.put("nonce\_str", WXPayUtil.*generateNonceStr*());//随机字符串  String url="https://api.mch.weixin.qq.com/pay/orderquery";  **try** {  String xmlParam = WXPayUtil.*generateSignedXml*(param, partnerkey);  HttpClient client=**new** HttpClient(url);  client.setHttps(**true**);  client.setXmlParam(xmlParam);  client.post();  String result = client.getContent();  Map<String, String> map = WXPayUtil.*xmlToMap*(result);  System.***out***.println(map);  **return** map;  } **catch** (Exception e) {  e.printStackTrace();  **return** **null**;  }  } |

### 1.2.3控制层

在portal-web的PayController.java新增方法

|  |
| --- |
| /\*\*  \* 查询支付状态  \* **@param** out\_trade\_no  \* **@return**  \*/  @RequestMapping("/queryPayStatus")  **public** Result queryPayStatus(String out\_trade\_no){  Result result=**null**;  **while**(**true**){  //调用查询接口  Map<String,String> map = weixinPayService.queryPayStatus(out\_trade\_no);  **if**(map==**null**){//出错  result=**new** Result(**false**, "支付出错");  **break**;  }  **if**(map.get("trade\_state").equals("SUCCESS")){//如果成功  result=**new** Result(**true**, "支付成功");  **break**;  }  **try** {  Thread.*sleep*(3000);//间隔三秒  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  **return** result;  } |

## 1.3检测支付状态-前端代码

### 1.3.1服务层

在payService.js新增方法

|  |
| --- |
| //查询支付状态  **this**.queryPayStatus=**function**(out\_trade\_no){  **return** $http.get('pay/queryPayStatus.do?out\_trade\_no='+out\_trade\_no);  } |

### 1.3.2控制层

在payController.js中新增方法

|  |
| --- |
| //查询支付状态  queryPayStatus=**function**(out\_trade\_no){  payService.queryPayStatus(out\_trade\_no).success(  **function**(response){  **if**(response.success){  location.href="paysuccess.html";  }**else**{  location.href="payfail.html";  }  }  );  } |

在createNative方法的回调方法中调用此查询方法

|  |
| --- |
| //本地生成二维码  $scope.createNative=**function**(){  payService.createNative().success(  **function**(response){  $scope.money= (response.total\_fee/100).toFixed(2) ; //金额  $scope.out\_trade\_no= response.out\_trade\_no;//订单号  //二维码  **var** qr = **new** QRious({  element:document.getElementById('qrious'),  size:250,  level:'H',  value:response.code\_url  });  queryPayStatus(response.out\_trade\_no);//查询支付状态  }  );  } |

## 1.4查询时间限制

### 1.4.1问题分析

如果用户到了二维码页面一直未支付，或是关掉了支付页面，我们的代码会一直循环调用微信接口，这样会对程序造成很大的压力。所以我们要加一个时间限制或是循环次数限制，当超过时间或次数时，跳出循环。

### 1.4.2代码完善

（1）修改portal-web工程PayController.java的queryPayStatus方法

|  |
| --- |
| @RequestMapping("/queryPayStatus")  **public** Result queryPayStatus(String out\_trade\_no){  Result result=**null**;  **int** x=0;  **while**(**true**){  //调用查询接口  .......  **try** {  Thread.*sleep*(3000);//间隔三秒  } **catch** (InterruptedException e) {  e.printStackTrace();  }  //为了不让循环无休止地运行，我们定义一个循环变量，如果这个变量超过了这个值则退出循环，设置时间为5分钟  x++;  **if**(x>=100){  result=**new** Result(**false**, "二维码超时");  **break**;  }  }  **return** result;  } |

（2）修改payController.js

|  |
| --- |
| //查询支付状态  queryPayStatus=**function**(out\_trade\_no){  payService.queryPayStatus(out\_trade\_no).success(  **function**(response){  **if**(response.success){  location.href="paysuccess.html";  }**else**{  **if**(response.message=='二维码超时'){  $scope.createNative();//重新生成二维码  }**else**{  location.href="payfail.html";  }  }  }  );  } |

# 2.品优购-支付日志

## 2.1需求分析

我们现在系统还有两个问题需要解决：

1. 系统中无法查询到支付记录
2. 支付后订单状态没有改变

我们现在就来解决这两个问题。

实现思路：

1. 在用户下订单时，判断如果为微信支付，就向支付日志表添加一条记录，信息包括支付总金额、订单ID（多个）、用户ID 、下单时间等信息，支付状态为0（未支付）
2. 生成的支付日志对象放入redis中，以用户ID作为key，这样在生成支付二维码时就可以从redis中提取支付日志对象中的金额和订单号。

（3）当用户支付成功后，修改支付日志的支付状态为1（已支付），并记录微信传递给我们的交易流水号。根据订单ID（多个）修改订单的状态为2（已付款）。

## 2.2表结构分析

**tb\_paylog 支付日志表**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 长度 | 含义 |
| out\_trade\_no | varchar | 30 | 支付订单号 |
| create\_time | datatime |  | 创建时间 |
| pay\_time | datatime |  | 支付完成时间 |
| total\_fee | bigint |  | 支付金额（分） |
| transaction\_id | varchar | 30 | 交易流水号 |
| trade\_state | varchar | 1 | 交易状态 |
| pay\_type | varchar | 1 | 支付类型：  1:微信  2:支付宝  3:网银 |
| order\_list | varchar | 200 | 订单表ID串，用逗号分隔 |

## 2.3插入日志记录

修改buyer-service工程OrderServiceImpl.java 的add方法。

内容：判断如果支付方式为微信支付，向数据库插入支付日志记录，并放入redis存储

|  |
| --- |
| @Autowired  **private** OrderDao orderDao;  @Autowired  **private** OrderItemDao orderItemDao;  @Autowired  **private** RedisTemplate redisTemplate;  @Autowired  **private** CartService cartService;  @Autowired  **private** PayLogDao payLogDao;    @Autowired  **private** IdWorker idWorker;  //保存订单  **public** **void** add(Order order){  List<Cart> cartList = cartService.  findCartList((List<Cart>) redisTemplate.boundHashOps("CART").get(order.getUserId()));  //订单ID集合  List<String> ids = **new** ArrayList<>();  //所有订单的总金额  **double** totalFee = 0;      **for** (Cart cart : cartList) {  //订单编号:  **long** id = idWorker.nextId();  ids.add(String.*valueOf*(id));  order.setOrderId(id);  //支付类型: 1: 在线支付(只包含微信支付) 2: 货到付款 页面选择中已经有了  //支付状态  order.setStatus("1");  //不要邮费  order.setPostFee("0");  //来源  order.setSourceType("2");//2:PC  //创建时间  order.setCreateTime(**new** Date());  //更新时间  order.setUpdateTime(**new** Date());  //商家ID  order.setSellerId(cart.getSellerId());  //实付金额  **double** money = 0;  List<OrderItem> orderItemList = cart.getOrderItemList();  **for** (OrderItem orderItem : orderItemList) {  //订单详情表  //详情表主键  orderItem.setId(idWorker.nextId());  //外键  orderItem.setOrderId(id);  money += orderItem.getTotalFee().doubleValue();//金额累加  orderItemDao.insertSelective(orderItem);  }  totalFee += money;  //实付金额  order.setPayment(**new** BigDecimal(money));    //保存订单  orderDao.insertSelective(order);  }    //如果是微信支付  **if**("1".equals(order.getPaymentType())){  //保存支付日志 表  PayLog payLog = **new** PayLog();  //支付订单号  payLog.setOutTradeNo(String.*valueOf*(idWorker.nextId()));  //创建时间  payLog.setCreateTime(**new** Date());  //支付完成时间  //支付金额  payLog.setTotalFee((**long**)totalFee);  //交易流水号 暂无  // 交易状态  payLog.setTradeState("0");  // 支付类型  payLog.setPayType("1");  //订单结果集  payLog.setOrderList(ids.toString().replace("[", "").replace("]", ""));  //用户ID  payLog.setUserId(order.getUserId());  payLogDao.insertSelective(payLog);  //将支付日志对象数据 保存缓存一份  redisTemplate.boundHashOps("payLog").put(order.getUserId(), payLog);  }    //删除缓存中的购物车  redisTemplate.boundHashOps("CART").delete(order.getUserId());      } |

## 2.4读取支付日志

### 2.4.1服务实现层

buyer-service的PayServiceImpl.java实现方法

|  |
| --- |
| @Autowired  **private** RedisTemplate redisTemplate;    //生成二维码  **public** Map<String,String> createNative(String name){  PayLog payLog = (PayLog) redisTemplate.boundHashOps("payLog").get(name);      String url = "https://api.mch.weixin.qq.com/pay/unifiedorder";  Map<String,String> param = **new** HashMap<>();  // 字段名 变量名 必填 类型 示例值 描述  // 公众账号ID appid 是 String(32) wxd678efh567hg6787 微信支付分配的公众账号ID（企业号corpid即为此appId）  param.put("appid", appid);  ……………………………………………………..  //总金额  map.put("total\_fee", payLog.getTotalFee().toString());  //订单号  map.put("out\_trade\_no", payLog.getOutTradeNo());  //生成二维码的URL  map.put("code\_url", xmlToMap.get("code\_url"));  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }    **return** map; |

### 2.4.3控制层

修改portal-web工程PayController.java的createNative方法

实现思路：调用获取支付日志对象的方法，得到订单号和金额

|  |
| --- |
| @Reference  **private** PayService payService;    //生成二维码  @RequestMapping("/createNative")  **public** Map<String,String> createNative(){  String name = SecurityContextHolder.*getContext*().getAuthentication().getName();  **return** payService.createNative(name);  } |

## 2.5修改订单状态

### 2.5.1服务实现层

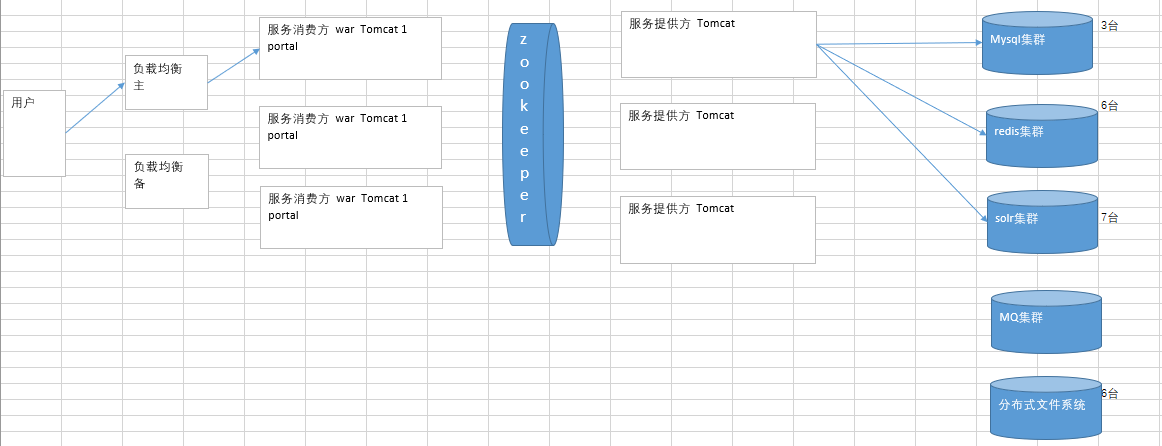
在buyer-service工程OrderServiceImpl.java实现该方法.

这个方法主要做三件事：

1. 修改支付日志状态
2. 修改关联的订单的状态
3. 清除缓存中的支付日志对象

|  |
| --- |
| @Autowired  **private** PayLogDao pagLogDao;  @Autowired  **private** OrderDao orderDao;  @Value("${appid}")  **private** String appid;  @Value("${partner}")  **private** String partner;  @Value("${partnerkey}")  **private** String partnerkey;  @Value("${notifyurl}")  **private** String notifyurl;  //查询订单是否支付成功  @Override  **public** Map<String, String> queryPayStatus(String out\_trade\_no,String name) {  // **TODO** Auto-generated method stub  String url = "https://api.mch.weixin.qq.com/pay/orderquery";  Map<String, String> param = **new** HashMap<>();  // 请求参数  // 字段名 变量名 必填 类型 示例值 描述  // 公众账号ID appid 是 String(32) wxd678efh567hg6787 微信支付分配的公众账号ID（企业号corpid即为此appId）  param.put("appid", appid);  // 商户号 mch\_id 是 String(32) 1230000109 微信支付分配的商户号  param.put("mch\_id", partner);  // 微信订单号 transaction\_id 二选一 String(32) 1009660380201506130728806387 微信的订单号，建议优先使用  // param.put("transaction\_id", WXPayUtil.generateUUID());  // 商户订单号 out\_trade\_no String(32) 20150806125346 商户系统内部订单号，要求32个字符内，只能是数字、大小写字母\_-|\*@ ，且在同一个商户号下唯一。 详见商户订单号  param.put("out\_trade\_no", out\_trade\_no);  // 随机字符串 nonce\_str 是 String(32) C380BEC2BFD727A4B6845133519F3AD6 随机字符串，不长于32位。推荐随机数生成算法  param.put("nonce\_str", WXPayUtil.*generateNonceStr*());  // 签名 sign 是 String(32) 5K8264ILTKCH16CQ2502SI8ZNMTM67VS 通过签名算法计算得出的签名值，详见签名生成算法  // 签名类型 sign\_type 否 String(32) HMAC-SHA256 签名类型，目前支持HMAC-SHA256和MD5，默认为MD5  **try** {  String xmlParam = WXPayUtil.*generateSignedXml*(param, partnerkey);  //发送  HttpClient httpClient = **new** HttpClient(url);  httpClient.setHttps(**true**);  httpClient.setXmlParam(xmlParam);  httpClient.post();  String result = httpClient.getContent();  System.***out***.println(result);  Map<String, String> map = WXPayUtil.*xmlToMap*(result);  **if**("SUCCESS".equals(map.get("trade\_state"))){  //已付款  //更新  //这个方法主要做三件事：  //1. 修改支付日志状态  //2. 修改关联的订单的状态  //3. 清除缓存中的支付日志对象  updateOrderStatus(map.get("transaction\_id"),name);  }  **return** map;  } **catch** (Exception e) {  // **TODO** Auto-generated catch block  // e.printStackTrace();  **return** **null**;  }    }  //更新  //这个方法主要做三件事：  //1. 修改支付日志状态  //2. 修改关联的订单的状态  //3. 清除缓存中的支付日志对象  **public** **void** updateOrderStatus(String transaction\_id,String name){  //1. 修改关联的订单的状态  PayLog payLog = (PayLog) redisTemplate.boundHashOps("payLog").get(name);  String[] orderIds = payLog.getOrderList().split(",");  Order order = **new** Order();  //付款状态  order.setStatus("2");  //更新时间  order.setUpdateTime(**new** Date());  //付款时间  order.setPaymentTime(**new** Date());  **for** (String id : orderIds) {  order.setOrderId(Long.*parseLong*(id));  orderDao.updateByPrimaryKeySelective(order);  }  //2. 修改支付日志状态  //付款时间  payLog.setPayTime(**new** Date());  //交易流水  payLog.setTransactionId(transaction\_id);  //付款状态 未付 已付  payLog.setTradeState("1");  pagLogDao.updateByPrimaryKeySelective(payLog);    //3. 清除缓存中的支付日志对象  redisTemplate.boundHashOps("payLog").delete(payLog.getUserId());  } |

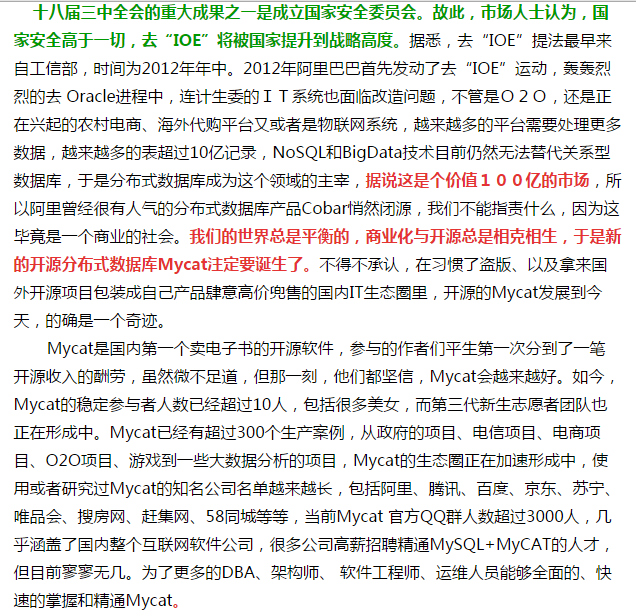
# 3.企业整体架构分析



# 4.Mycat中间件

## 4.1.Mycat介绍





## 4.2.Mycat分片原理



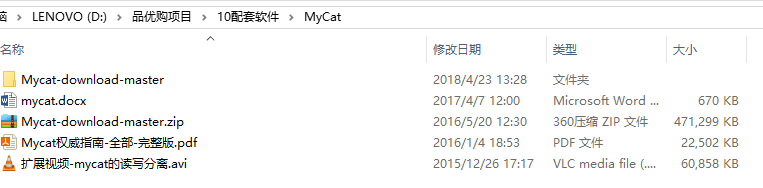
### 4.2.1.添加数据

|  |
| --- |
|  |

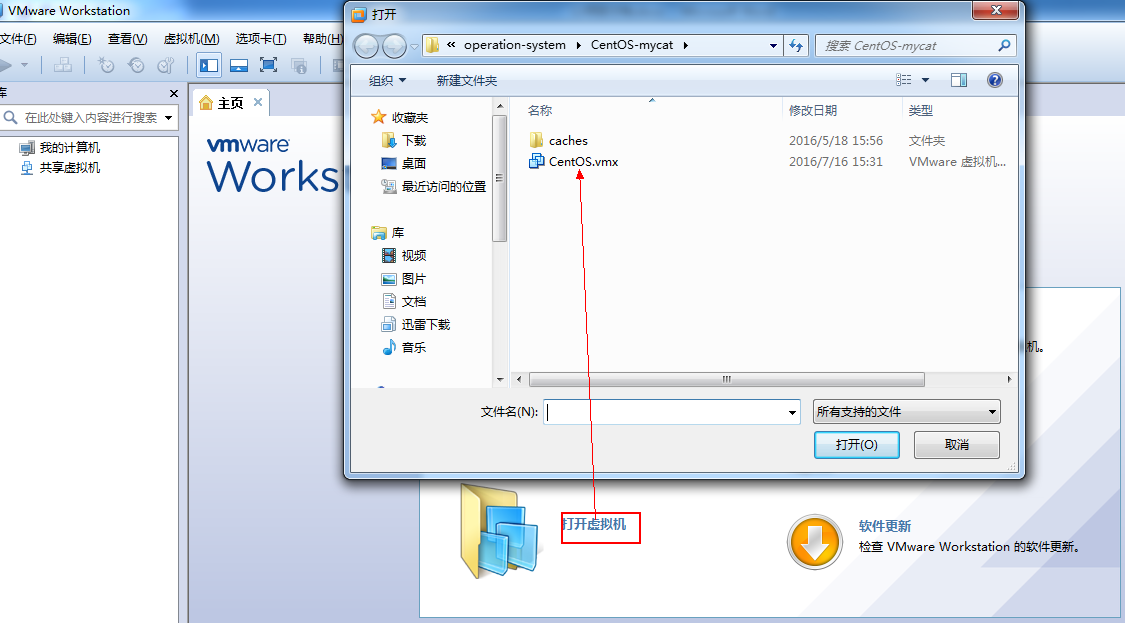
### 4.2.2.检索数据

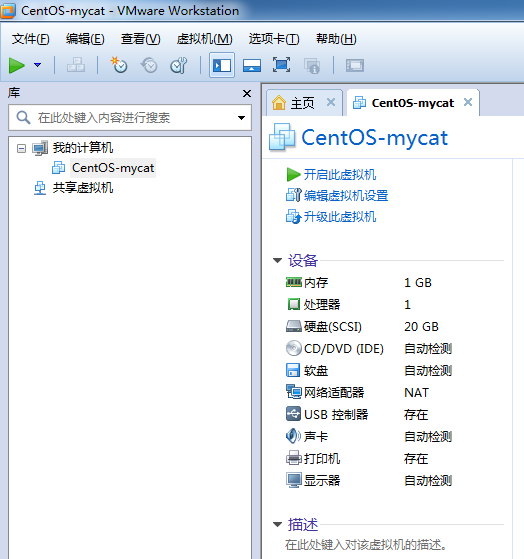
|  |
| --- |
|  |

## 4.3.Mycat资料



## 4.4.VMware中打开Centos-mycat





## 4.5.启动Mysql

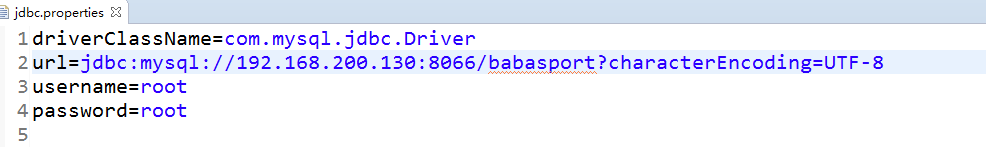
service mysqld start

## 4.6.启动mycat

[root@localhost ~]#cd mycat/bin

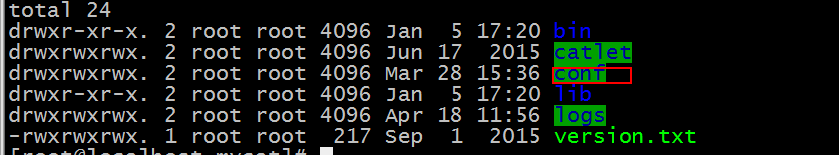
[root@localhost ~]#./mycat start

## 4.7.项目中连接Mycat

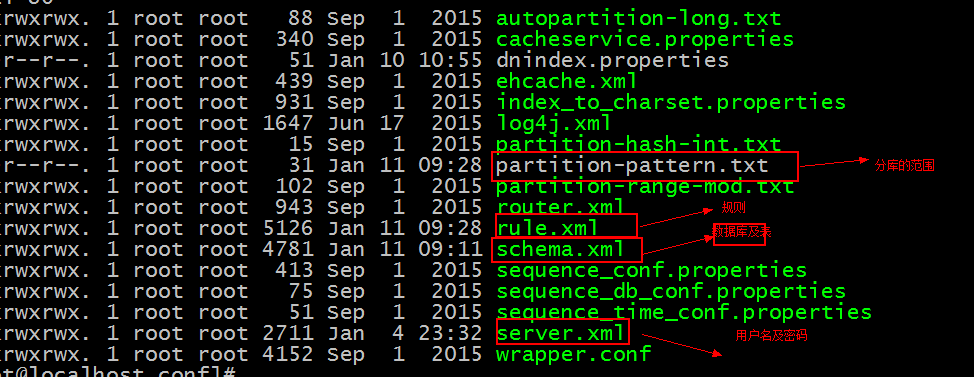


# 5.Mycat配置

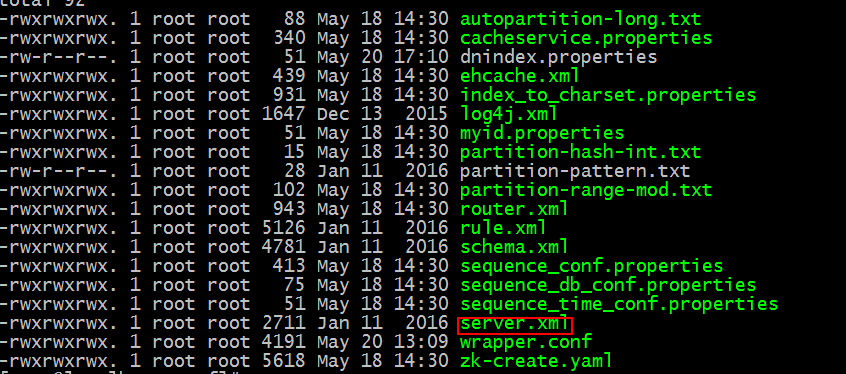
## 5.1.位置

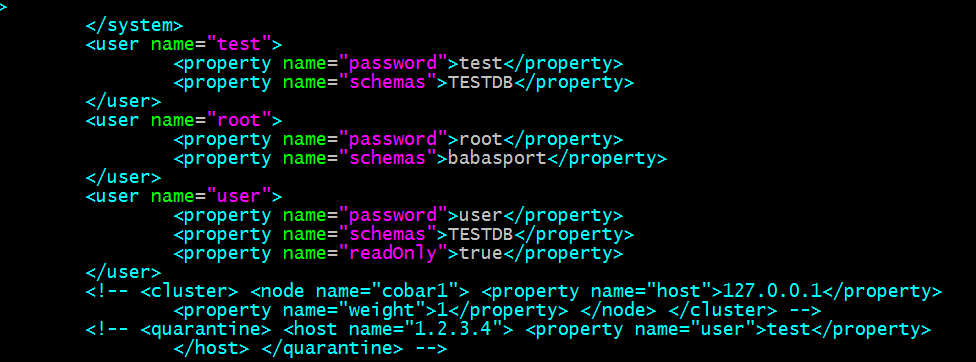


## 5.2.Mycat的配置文件

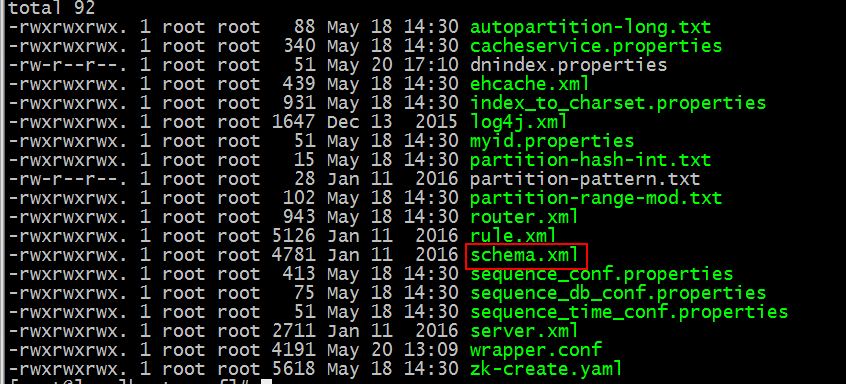


## 5.3.配置Mycat的用户名、密码

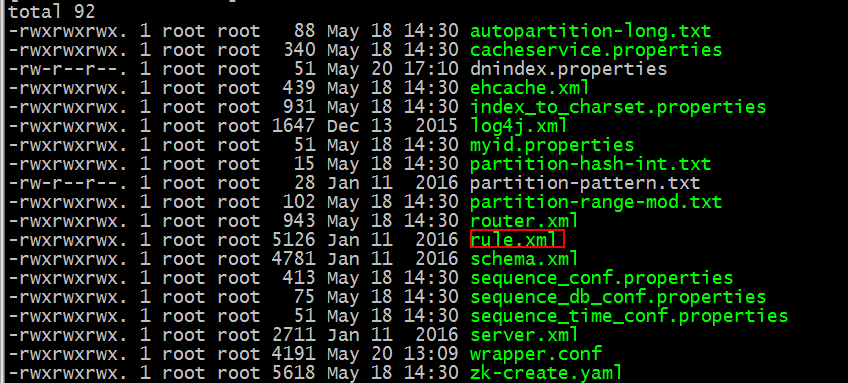


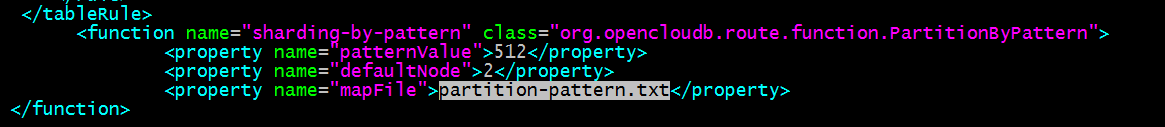


## 5.4.配置21张表



## 5.5.表的规则





## 5.6.Partition-pattern.txt

